An algebra for branching processes

David Delfieu, Médésu Sogbohossou

An algebra for branching processes

David Delfieu
IRCCyN
1, rue de Noe
BP 92101, France
david.delfieu@irccyn.ec-nantes.fr

Médésu Sogbohossou
LETIA
University of Abomey-Calavi
01 BP 2009 Cotonou, BENIN
sogbohossou_medesu@yahoo.fr

***Abstract*—The unfolding process of Petri Nets produces a set of causal nets where nodes are conditions or events and arcs express relations of causality, conflict or concurrency called branching processes. We propose in this paper an algebra and reduction rules allowing to extract informations, relation on events and a canonic representation of branching processes.**

I. Introduction

Petri nets are a modeling tool widely used in the formal study of concurrent discrete-event systems. Verification of properties is based on a computable state graph [1] if the net is bounded. For a highly concurrent system, this computation is hindered by a combinatory explosion. One cause of this explosion is the semantics of interleaving used to approximate the concurrency aspects of a system under modeling. Technics of unfolding [4]–[7] produce a-cyclic nets keeping concurrent aspects with partial order semantics.

Unfolding Petri nets give a set of causal nets where nodes are conditions or events. The arcs specify the causality relations. The exhibition of conflicts allows to partition this set in term of branching process. This paper proposes an algebra dedicated to the extraction of semantics in branching processes. This formal framework is based on a few basic assumptions and concepts and proposes a logical growth from expressions (well formed formulae) on terms to derivation rules and laws. In figure 1 is an unfolding. The events e_i stand for events and b_i for conditions:

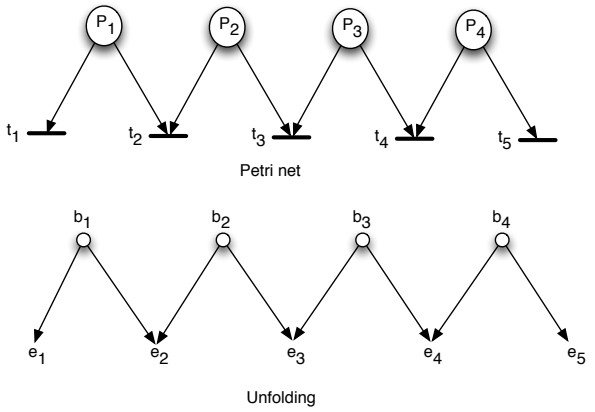

Fig. 1. Unfolding

The algebra presented in this paper offers a syntax to express this unfolding and the reduction rules proposed in this paper, determine all the maximal processes of the unfolding. By example, which are expressed by the following expressions:

$$(e_1 \oplus e_3 \oplus e_5) \perp (e_1 \oplus e_4) \perp (e_2 \oplus e_4) \perp (e_2 \oplus e_5)$$

$$(e_1 \oplus e_3) \perp e_2$$

Where $\perp$ stands for the exclusion and $\oplus$ aggregates events in a processus, whereas $e_1 \oplus e_3 \oplus e_5$ is a compact representation of $e_1 \oplus \overline{e_2} \oplus e_3 \oplus \overline{e_4} \oplus e_5$.

The originality of this approach is that it allows to extract non intuitive causalities from this set of events in conflicts, for example (figure 1), we will exhibit:

- Strong causalities: $e_3 \preccurlyeq (e_1 \oplus e_5)$ and $e_4 \preccurlyeq (e_1 \perp e_2)$, or $e_6 \preccurlyeq e_8$
- Weak causalities $e_1 \precsim e_3$, $e_1 \precsim e_5$,...

II. Related works

A process algebra (CCS) has been already developed by Milner [10]. CCS is based on two central ideas: The notion of observability and the concept of synchronized communication. The elements of the alphabet are observable events and concurrent systems (processes) can be specified with the use of three operators: sequence, choice and parallelism.

A main property of CCS is the rejection of distributivity of the sequence upon the choice. This rejection is based on the fact that $a.(x+y)$ and $a.x+a.y$ are "language equivalent" but they are not equivalent in term of behavior.

Let's consider the expression $x + y$. It represents a choice: either x or either y can be observed. Let's now consider the expression $a.(x + y)$, after the observation of a, $(x + y)$ stay observable which means that either x or either y can be observable. But, in the expression $a.x + a.y$ the distributivity has made a choice and this expression expresses two scenarios: one with a followed by x and one followed by y.

In CCS, Milner defines a finer law, the behavior law rejecting the distributivity of the sequence on the choice:

$$a.(x + y) \not\equiv_{behaviorally} a.x + a.y$$

In the context of unfolding, branching processes differ from the concurrents systems of Milner. Branching process are

constituted with occurred events. The notion of observability loses sense in term of realized process. An unfolding produces all the prefixes (a is a prefix) called branching processes, $a.x$ and $b.x$ are the maximal processes for the considered expression.

The distributivity is then kept valid and $a \preccurlyeq (x \oplus y)$ can be reduced in $(a \preccurlyeq x) \oplus (a \preccurlyeq y)$ where $a \preccurlyeq x$ and $a \preccurlyeq y$ stands for two (distinct) branching processes. This last expression asserts that a causes x and a causes y. Unfolding can be represented with a set of a-cyclic graph. In an unfolding, every arc carries a unique label because it identifies an event which represent is the firing of a transition of the underlying Petri net. e_1 represents the firing of a transition in a given state corresponding to a particular marking. Another label e_2 can represent the firing of the *same* transition with different state of the system (marking).

Branching process does not feet with process algebra on numerous other aspects. For example a difference can be noticed about parallelism. While unfolding keeps true parallelism, process algebra considers a parallelism as interleavings.

Another difference is relative to events and conditions which are nodes of different nature in an unfolding. Conditions and events differ in term of ancestor. Every condition is produced by at most one event ancestor (none for the condition standing for m_0, the initial marking), whereas every event may have $0, 1$ or n condition ancestor(s). In CCS, there is no distinction between conditions and events and process can be cyclic.

However the works developed in CCS have shown the interest of an algebraic formalization: it allows to extract information, to propose reduction rules, and it defines equivalences. And this proposed algebra is dedicated to branching process tends to yield the same objective than CCS:

- a syntax that expresses in non ambiguous way the relations between the events ;
- equivalence and reduction rules ;
- extraction of semantics ;
- a canonical form of an unfolding.

For example, in the figure 1, e_1, e_3 and e_5 are bounded by a particular relation. If two of these, for example e_1 and e_5 are in a process then e_3 must belong also, to the same process. This particular relation is captured by properties of $\oplus$ operator. In particular, we will establish that in a following theorem (VI):

$$(e_1 \oplus e_5) \perp (e_1 \oplus e_3 \oplus e_5) \equiv (e_1 \oplus e_3 \oplus e_5)$$

The second section exposes the basics notions about Petri net and unfolding. The third section depicts the new algebra, which is the contribution of this paper. The fourth section illustrates the use of the algebra on the unfolding of a non safe Petri Net.

III. Unfolding a Petri net

A. Petri net

A Petri net [9] $\mathcal{N} =< \mathcal{P}, \mathcal{T}, \mathcal{W} >$ is a triple with: $\mathcal{P}$, a finite set of places, $\mathcal{T}$, the finite set of transitions ($\mathcal{P} \cup \mathcal{T}$ are nodes of the net; ($\mathcal{P} \cap \mathcal{T} = \emptyset$), and $\mathcal{W} : (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P}) \longrightarrow \mathbb{N}$, the flow relation defining arcs (and their valuations) between nodes of $\mathcal{N}$. In the sequel, we will consider only *1-valued* Petri nets.

The pre-set (resp. post-set) of a node x is denoted ${}^\bullet x = \{y \in \mathcal{P} \cup \mathcal{T} \mid \mathcal{W}(y, x) > 0\}$ (resp. $x^\bullet = \{y \in \mathcal{P} \cup \mathcal{T} \mid \mathcal{W}(x, y) > 0\}$).

A marking of a Petri net $\mathcal{N}$ is a mapping $m : \mathcal{P} \longrightarrow \mathbb{N}$. A transition $t \in T$ is said *enabled* by m iff: $\forall p \in {}^\bullet t,\ m(p) \geq \mathcal{W}(p, t)$. This is denoted: $m \xrightarrow{t}$. Firing of t leads to the new marking m' ($m \xrightarrow{t} m'$): $\forall p \in \mathcal{P},\ m'(p) = m(p) - \mathcal{W}(p, t) + \mathcal{W}(t, p)$. The initial marking is denoted m_0.

A Petri net is *k-bounded* iff $\forall m$, reachable from $m_0, m(p) \leq k$ (with $p \in \mathcal{P}$). It is said *safe* when *1-bounded*. Two transitions are in a *structural conflict* when they share at least one pre-set place; a conflict is *effective* when these transitions are both enabled by a same marking.

B. Unfolding

In [5], the notion of *branching process* is defined as an initial part of a run of a Petri net respecting its partial order semantics and possibly including non deterministic choices (conflicts). It is another Petri net that is acyclic and the largest branching process of an initially marked Petri net is called *the* unfolding of this net. Resulting net from an unfolding is a labeled occurrence net, a Petri net whose places are called *conditions* (labeled with their corresponding place name in the original net) and transitions are called *events* (labeled with their corresponding transition name in the original net).

An occurrence net [3] $\mathcal{O} =< \mathcal{B}, \mathcal{E}, \mathcal{F} >$ is a 1-valued arcs Petri net, with $\mathcal{B}$ the set of conditions, $\mathcal{E}$ the set of events, and $\mathcal{F}$ the flow relation (1-valued arcs), such that: $|{}^\bullet b| \leq 1$ ($\forall b \in \mathcal{B}$), ${}^\bullet e \neq \emptyset$ ($\forall e \in \mathcal{E}$), and $\mathcal{F}^+$ (the transitive closure of $\mathcal{F}$) is a strict order relation, from which $\mathcal{O}$ is acyclic. $Min(\mathcal{O}) = \{b \mid b \in \mathcal{B}, |{}^\bullet b| = 0\}$ is the minimal conditions and corresponds to the initial marking. Also, $Max(\mathcal{O}) = \{x \mid x \in \mathcal{B} \cup \mathcal{E}, |x^\bullet| = 0\}$ are maximal nodes.

Three kinds of relations could be defined between the nodes of an occurrence net. First, two nodes $x, y \in \mathcal{B} \cup \mathcal{E}$ are in *strict causal relation* if $(x, y) \in \mathcal{F}^+$, and will be denoted $x \preccurlyeq y$. $\forall b \in \mathcal{B}$, if $e_1, e_2 \in b^\bullet$ ($e_1 \neq e_2$), then e_1 and e_2 are in *conflict relation*, denoted $e_1 \sharp e_2$. More generally, for $x, y \in \mathcal{B} \cup \mathcal{E}$, if $e_1 \preccurlyeq x$ et $e_2 \preccurlyeq y$, then $x \sharp y$, $x \sharp e_2$ and $e_1 \sharp y$. Symbol $\wr$ represents *concurrency relation*: $x \wr y$ iff $\neg((x \preccurlyeq y) \vee (y \preccurlyeq x) \vee (x \sharp y))$.

A *cut* is a set of conditions all in mutually concurrency relation. A *B-cut* is a maximal cut according to inclusion and represents a marking of the original net.

The *local configuration* of an event e, denoted $[e]$, is the set of events e' such that $e' \preccurlyeq e$ (relation $\prec$ is defined on $\mathcal{F}^*$ where $\mathcal{F}^*$ is the transitive and reflexive closure of $\mathcal{F}$).

Let be a net $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{W})$. $< \mathcal{N}, m_0 >$ admits $Unf =< \mathcal{O}, \lambda >$ as branching process (or unfolding) if:

- $\mathcal{O} =< \mathcal{B}, \mathcal{E}, \mathcal{F} >$ is an occurrence net;
- λ is the labeling function such that $\lambda : \mathcal{B} \cup \mathcal{E} \to \mathcal{P} \cup \mathcal{T}$. It verifies the following properties:

- $\lambda(\mathcal{B}) \subseteq \mathcal{P}$, $\lambda(\mathcal{E}) \subseteq \mathcal{T}$;
- $\lambda(Min(\mathcal{O})) = m_0$;
- for all $e \in \mathcal{E}$, restriction of λ to ${}^\bullet e$ (resp. $e^\bullet$) is a bijection between ${}^\bullet e$ (resp. $e^\bullet$) and ${}^\bullet\lambda(e)$ (resp. $\lambda(e)^\bullet$). We have ${}^\bullet\lambda(e) = \lambda({}^\bullet e)$ and $\lambda(e)^\bullet = \lambda(e^\bullet)$, what means that λ preserves the environment of a transition.

The resulting labeled occurrence net is defined up to isomorphism, what means that we obtain a same structure of net up to the name of nodes, and with same labeling of nodes.

A *causal net* $\mathcal{C}$ is an occurrence net $\mathcal{C} =< \mathcal{B}, \mathcal{E}, \mathcal{F} >$ with an adding restriction: $\forall b \in \mathcal{B},\ |b^\bullet| \leq 1$. So, all conflicts are resolved in a causal net, and this net corresponds to a partial order run in an unfolding. The set of events of a causal net is called a *process*. Notice that only the order relations $\prec$ and $\wr$ are admitted between events of a process.

IV. Definition of the algebra

We have shown in the previous section, how unfolding exhibits causal nets and conflicts. Otherwise, every couple of events which are not bounded by a causal relation or the same conflict set are in concurrency. We have developed a software producing the unfolding which builds a table making explicit every binary relations between events. In fact, this table establishes only the relations of causality and exclusion. If a binary relation is not explicit in the table, it means that the couple of events are in a concurrency relation. We show in this section that this table has led us to introduce a new operator which collapses events in a process.

Let $\mathcal{EB} = \mathcal{E} \cup \mathcal{B}$ a finite alphabet, composed of the events and the conditions generated by the unfolding. In this set every couple of elements satisfies:

- a causality relation $\mathcal{C}$
- a concurrency relation $\mathcal{I}$
- an exclusive relation $\mathcal{X}$

The set of binary relations is partitioned in three parts as illustrated in figure 2:

This last figure makes appear the operators of this algebra. A couple of events can be bounded by a relation of causality, exclusion or concurrency. The causality and concurrency set can be collapsed to constitute the set of all the processes of an unfolding. And this set is the complementary of the exclusion set. If some events are not in a relation of exclusion, they are in the same process. To express the fact that events are in the same process, a new operator noted $\oplus$ is introduced.

The negation operator allows to switch from exclusion set to the process set. The algebra is then defined by

$$\mathcal{A} = \{\mathcal{EB}, (\preccurlyeq\ ,\ \wr\ ,\ \perp\ ,\ ^-, \oplus)\}$$

$\oplus$ and $\perp$ are almost similar to boolean operator "." and "$+$", except that boolean operators are mutually distributive and we will show in the section IV-B2, that $\perp$ is not distributive over $\oplus$.

The internal composition laws are:

- $\preccurlyeq$: The causality ;

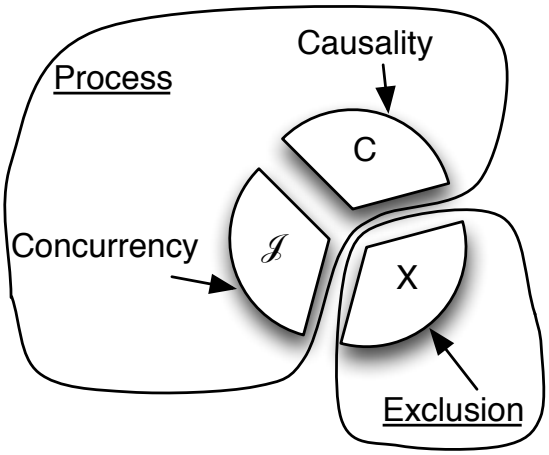

Fig. 2. Partition of binary relations

- $\wr$: The concurrency ;
- $\perp$: The exclusion ;
- $^-$: The negation ;
- $\oplus$: building process.

A. Definitions of operators

In the following, e_1, e_2, e_3 are elements of $\mathcal{EB}$.

1) Causality: $\mathcal{C}$ is the set of all the causalities $\mathcal{C}$ between every elements of $\mathcal{EB}$. $e_1 \preccurlyeq e_2$ if e_2 is in the local configuration of e_1:

$$\models e_1 \preccurlyeq e_2 \text{ if } e_2 \in [e_1]$$

Obviously, if $e_1 \prec e_2$ then $e_1 \preccurlyeq e_2$. The occurrence of e_1 is a necessary condition to the occurrence of e_2.

Properties:

- $\preccurlyeq$ is associative and transitive;
- $\preccurlyeq$ has a neutral element $t \preccurlyeq e \equiv e$;
- every element of $\mathcal{EB}$ has an opposite: $f \preccurlyeq e \equiv \bar{e}$.

2) Exclusion: $\mathcal{X}$ is the set of all the exclusion relations between every elements of $\mathcal{EB}$. To define the this operator let's first go back to it's definition given in the section concerning the unfolding:

$$e_1 \sharp\ e_2 \equiv ((\bullet e_1 \cap \bullet e_2 \neq \emptyset) \text{ and } (e_1 \neq e_2)$$

The definition of $\perp$ is more generally: two events are in exclusion *iff* they are either in direct conflict, either it exists a conflict at the level of a common ancestor. And so they belongs to different processes:

$$e_1 \perp e_2 \equiv ((\bullet e_1 \cap \bullet e_2 \neq \emptyset) \text{ or } (\exists e_i, e_i \preccurlyeq e_2 \text{ and } e_1 \perp e_i))$$

Properties:

- Commutativity: $e_1 \perp e_2 \equiv e_2 \perp e_1$
- Neutral element: $e \perp f \equiv e$
- Absorbing element: $e \perp t \equiv t$

Let's note that $\perp$ is not transitive.

In the previous example (fig. 1): $\models e_1 \perp e_2$, $\models e_2 \perp e_3$, but $\models e_1 \wr e_3$

We will use a pre-fixed notation for an n-ary use of $\perp$ $(n > 2)$:

$$(\perp \ e_1 \ e_2 \ e_3 \ e_4 \ ...)$$

which abbreviates $p \models (\perp \ e_1 \ e_2 \ e_3 \ e_4 \ ...)$ and which abreviates the following set of exclusions:

$$p \models (e_1 \perp e_2),\ p \models (e_2 \perp e_3),\ p \models (e_3 \perp e_4)...$$

3) Concurrency: $\mathcal{I}$ is the set of every couple of element of $\mathcal{EB}$ in concurrency. e_1 and e_2 are in concurrency if the occurrence of one is independent of the occurrence of the other. So, $e_1 \wr e_2$ iff e_1 and e_2 are neither in causality neither in exclusion.

$$e_1 \wr e_2 \equiv \neg((e_1 \perp e_2) \text{ or } (e_1 \preccurlyeq e_2) \text{ or } (e_2 \preccurlyeq e_1))$$

Properties:

- $\wr$ is commutative and associative;
- $\wr$ is not transitive;

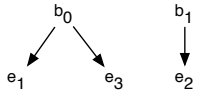

Fig. 3. Non-transitivity of $\wr$

The figure (fig. 3) illustrates a counter-example:

- e_2 is an concurrent event, so:
  $e_1 \wr e_2$
  $e_2 \wr e_3$
- but $e_1 \perp e_3$

- Neutral element: $e \wr t \equiv e$
- Absorbing element: $e \wr f \equiv f$

In the same manner used for $\perp$, we will use a pre-fixed notation for $\wr$ for an n-ary use $(n > 2)$:

$$(\wr \ e_1 \ e_2 \ e_3 \ e_4 \ ...)$$

which abbreviates the set of concurrencies:

$$(e_1 \wr e_2),\ (e_2 \wr e_3),\ (e_3 \wr e_4)...$$

4) Process: The figure fig. 2) illustrates that $\oplus$ aggregates events in one process. Two events e_1 and e_2 are in the same process if e_1 causes e_2 or if e_1 is concurrent with e_2:

$$e_1 \oplus e_2 \equiv (e_1 \preccurlyeq e_2) \text{ or } (e_1 \wr e_2)$$

The meaning of this operator is similar to the linear connector $\oplus$ of MILL [2] in the sense that its allows to aggregates resources. But in the context of unfolding, events or conditions are unique and then they cannot be counted. Thus, this operator is idempotent.

The relation $e_1 \oplus e_2$ defines that e_1 and e_2 are in the same process. This operator is n-ary. In n-ary expressions we will use the pre-fixed notation $(\oplus e_1 e_2 e_3 ...)$ which abbreviates $(e_1 \oplus e_2)$ and $(e_2 \oplus e_3)$....

Properties:

- $\oplus$ is commutative, associative and transitive;
- Idempotency: $e \oplus e \equiv e$
- Neutral element: $e \oplus t \equiv e$
- Absorbing element: $e \oplus f \equiv f$
- $e \oplus \bar{e} \equiv t$

In the same manner used for $\perp$, we will use a pre-fixed notation for $\oplus$ for an n-ary use $(n > 2)$: $(\oplus \ e_1 \ e_2 \ e_3 \ e_4 \ ...)$ which abbreviates the set of concurrencies: $(e_1 \oplus \ e_2)$, $(e_2 \oplus e_3)$, $(e_3 \oplus \ e_4)$ ...

B. Relations between operators

The distributivities over $\perp$ are used in the transformation of an expression in the canonical form. The other distributivities will be used in the reduction process.

1) Distributivities:

- $\preccurlyeq$ est distributive over $\wr$.
- $\perp$ is distributive $\wr$.
- $\oplus$ is distributive over $\perp$[1] and $\wr$.
- $\wr$ is distributive over $\perp$ and $\oplus$.

2) Non-distributivities:

- $\oplus$, $\perp$ and $\wr$ are non distributive over $\prec$:

$$e \oplus (e_1 \preccurlyeq e_2) \not\equiv (e \oplus e_1) \preccurlyeq (e \oplus e_2)$$

$$e \perp (e_1 \preccurlyeq e_2) \not\equiv (e \perp e_1) \preccurlyeq (e \perp e_2)$$

$$e \wr (e_1 \preccurlyeq e_2) \not\equiv (e \wr e_1) \preccurlyeq (e \wr e_2)$$

For the first and the second item, the right member these items are of the form: $(e \ op \ e_1) \preccurlyeq (e \ op \ e_2)$ (with $op = \oplus, \perp \ \wr$). Theses expressions establish a temporal precedence between $(e \ op \ e_1)$ over $(e \ op \ e_2)$ which does not exist in the left member. For example, in the second member of the first "non distribitivity", $(e \oplus e_1)$ cannot precede $(e \oplus e_2)$ because of the presence of e in the two expressions.

$$e \perp (e_1 \oplus e_2) \not\equiv (e \perp e_1) \oplus (e \perp e_2)$$

if $e \perp (e_1 \oplus e_2)$ implies $(e \perp e_1) \oplus (e \perp e_2)$ the reciprocal is false. This is a semi-distributivity.

- $\preccurlyeq$ is non distributive over $\oplus$:

$$e \preccurlyeq (e_1 \oplus e_2) \not\equiv (e \preccurlyeq e_1) \oplus (e \preccurlyeq e_2)$$

Because $(e \preccurlyeq e_1) \oplus (e \preccurlyeq e_2)$ can derive in $e \preccurlyeq (e_1 \perp e_2)$.

To resume the distributivities ($\surd$) and the non distributivities ($\times$) are expressed in the following table. The distributivities must be read from in rows. For example, the first line expresses that $\preccurlyeq$ is only distributive over $\perp$.

[1]The distributivity over $\perp$ is rejected in CCS

	$\preccurlyeq$	$\perp$	$\oplus$	$\wr$
$\preccurlyeq$		$\times$	$\times$	$\surd$
$\perp$	$\times$		$\times$	$\surd$
$\oplus$	$\times$	$\surd$		$\surd$
$\wr$	$\times$	$\surd$	$\surd$	

C. Axiom, derivation and theorem

1) Axioms:

Axiom 1 ($\oplus$):

$$e_1 \oplus e_2 \equiv_{def} e_1 \bar{\perp} e_2$$

$$e_1 \bar{\oplus} e_2 \equiv_{def} e_1 \perp e_2$$

The following axiom explicits a conflict:

Axiom 2 (Exclusion):

$$e_1 \perp e_2 \equiv_{def} (\bar{e_1} \oplus e_2) \perp (e_1 \oplus \bar{e_2})$$

The following axiom expresses that a concurrency relation encompasses three alternative processes: the occurrence of e_1, the occurrence of e_2, or the process composed with e_1 and e_2:

Axiom 3 (Concurrency):

$$e_1 \wr e_2 \equiv_{def} e_1 \perp e_2 \perp e_1 \oplus e_2$$

2) Derivation Rules: Theses rules allow to reduct process:

1) Reduction of $\preccurlyeq$

$$\frac{\vdash e_1 \preccurlyeq e_2}{\vdash e_1 \oplus e_2} \; Caus$$

2) Modus Ponens:

$$\frac{\vdash e_1 \quad \vdash e_1 \preccurlyeq e_2}{\vdash e_2} \; MP$$

3) Dual form:

$$\frac{\vdash \bar{e_1} \quad \vdash e_1 \preccurlyeq e_2}{\vdash \bar{e_2}} \; MP'$$

4) General form:

$$\frac{\vdash \bar{e_1} \quad \vdash (e_1 \oplus e_2 \ldots \oplus e_n) \preccurlyeq e}{\vdash \bar{e}} \; GMP'$$

5)

$$\frac{\vdash (\oplus \; b \; ... \; E) \quad \vdash (\oplus \; b \; ...) \preccurlyeq (\perp \; e_1 \; e_2)}{\vdash (\perp \; (\oplus \; e_1 \; (\overline{e_2}) \; E)(\oplus \; e_2 \; \; (\overline{e_2})E))} \; D$$

This rule expresses that a conflict divides a process.

6)

$$\frac{\vdash \bar{e_1} \oplus E}{\vdash E} \; S$$

This rule is applied on canonical form to clear the negations.

V. Canonical form

We give first, the definitions of algebraic definition of unfolding and process:

A. Preliminar definitions

Definition 1 (AEU): An Algebraic Expression of Unfolding (AEU) is a well formed formulae inductively defined by:

- t stand for true, f stand for false are terms ;
- $\forall e \in \mathcal{EB}, e, \bar{e}$ are terms ;
- a term is an AEU
- if ϕ, ϕ_1, ϕ_2 are AEU:
  - $\phi_1 \oplus \phi_2$ is an AEU ;
  - $\phi_1 \perp \phi_2$ is an AEU ;
  - $\phi_1 \wr \phi_2$ is an AEU ;
  - $\phi_1 \preccurlyeq \phi_2$ is an AEU ;

Theorem 1: Every unfolding of Petri nets is an algebraic expression of unfolding.

Proof: As the unfolding process produces only three types of relation between events: causality, concurrency and exclusion, every unfolding can be translated in a AEU. As evoked in the figure 1, by definition, every pair of elements (e_1, e_2) in $\mathcal{EB}|_X$ - either $e1 \prec$ either $e_1 \wr e_2$ is bounded by the $\oplus$ operator. ■

An AEU is composed of processes.

Definition 2 (Process): A process is a well formed formulae inductively defined by:

- t, f are terms ;
- $\forall e \in \mathcal{EB}$ are terms ;
- a term is a process ;
- If $p_1, p_2, \ldots p_n$ are processes:
  - $p_1 \oplus p_2 \ldots \oplus p_n$ is a process ;
  - $p_1 \wr p_2 \ldots \wr p_n$ is a process ;
  - $p_1 \preccurlyeq p_2 \ldots \preccurlyeq p_n$ is a process ;

$\perp$ is an operator that partitions the AEU into processes.

B. Canonical form

Definition 3: A canonic process is a formula expressed on elements of $\mathcal{EB}$ and with the operators $\oplus$ and $\wr$ ordered by an alphanumeric sort on the name of its symbol.

Theorem 2 (Canonical form): Let's consider an unfolding U, this form can be reduced in the following form:

$$U = p_0 \perp p_1 \perp ... \perp p_n$$

This form is canonic and allows to extract all the canonic processes p_i.

Proof: In an unfolding every operator $\preccurlyeq$ can be reduced in $\oplus$ by deduction rule $Caus$ (see section IV-C2). Moreover, $\oplus$ is distributive on the other operators $\perp$ and $\wr$ and $\perp$ is distributive over $\wr$. So $\perp$ can be factorized in every sub-formula. In fine an alphanumeric sort on symbols of the process can then be applied to assure the unicity of the form. ■

Definition 4 (member of a canonic process): Let $e \in \mathcal{EB}$, ϕ a term, p a process

- $e \in p$ iff e is a term of the process p.
- $\phi \in p$ iff ϕ is a process and ϕ is a sub-formula of p.

Now we can define the membership of an unfolding. An element $e \in \mathcal{EB}$, e belongs to an unfolding U iff $\exists p_i \in U$ such that $e \in p_i$. Let's now define how a formula can be interpreted.

Let $U = p_0 \perp p_1 \perp ... \perp p_n$ be an unfolding, constituted by n processes, e an event, ϕ a term:

- $\models e$ if $e \in U$
- $p_i \models e$ if $e \in p_i$.
- $\models \bar{e}$ if $e \notin U$
- $p_i \models \bar{e}$ if $e \notin p_i$.
- $\models \phi$ if $\phi \in p_i, \forall p_i \in U$,
- $p_i \models \phi$ if $\phi \in p_i$.
- $\models \phi \perp \psi$ if

$$\exists p_i, p_j \in U \text{ such as } \begin{cases} p_i & \models \phi \\ p_j & \models \psi \end{cases}$$

The canonical form of the upper graph of the example of the figure 1:

$$\begin{array}{lll} (\perp\ e_1\ e_2\ e_3\ e_4\ e_5) \equiv (\perp & (\oplus\ e_1\ \overline{e_2}\ e_3\ \overline{e_4}\ e_5) \\ & (\oplus\ e_1\ \overline{e_2}\ \overline{e_3}\ e_4\ \overline{e_5}) \\ & (\oplus\ \overline{e_1}\ e_2\ \overline{e_3}\ e_4) \\ & (\oplus\ \overline{e_1}\ e_2\ \overline{e_3}\ e_5)) \end{array}$$

C. Extraction of semantics

The extraction of semantics is based on the formulation in canonical form and the analysis of membership of events in processes.

In the unfolding the causality ($\prec$) is defined by the transitive closure of $\mathcal{F}^+$, where $\mathcal{F}$ is the flow relation in causal nets which is a strict notion of causality, because it induces a path between nodes to be causal. We introduce here a larger notion of causality ($\preccurlyeq$) where two events can be causal event even if it does not exist a path between them.

Definition 5 (Strong causality): $e_1 \preccurlyeq e_2$ iff for every p_i in U, whenever e_1 belong to process p_i, e_2 belongs to p_i, else if e_1 does not belong to a process p_j then e_2 cannot belong to p_j.

In the previous example, $\forall p_i \in U$, whenever $p_i \models e_3$ then $p_i \models (e_1 \oplus e_5)$. Moreover, if $p_i \models e_4$ then $p_i \models (e_1 \perp e_2)$, in the same manner if $p_i \models e_5$ then $p_i \models (e_1 \perp e_2)$. So, we can deduce of the canonical form the following causalities:

- $e_3 \preccurlyeq (e_1 \oplus e_5)$
- $e_4 \preccurlyeq (e_1 \perp e_2)$
- $e_5 \preccurlyeq (e_2 \perp e_3)$
- $e_6 \preccurlyeq e_8$
- $e_8 \preccurlyeq e_6$

In this example, it is obvious that e_6 forces e_8 and vice versa. A weaker causality can be defined relaxing the second assertion in the definition of $\preccurlyeq$:

Definition 6 (Weak causality): $e_1 \precsim e_2$ iff for every p_i in U, whenever e_1 belong to process p_i, e_2 belongs to p_i, else if e_1 does not belong to a process p_j then e_2 *can* belong to p_j.

In the previous example (introduction), whenever $\forall p_i \in U$, if $p_i \models e_2$ then $p_i \models e_4$, but it exists a process $p_j = e_1 \oplus e_4$ where $p_j \models e_4$ but $p_j \not\models e_2$. In the same manner $\forall p_i \in U$, if $p_i \models e_3$ then $p_i \models e_5$, but it exists a process $p_j = e_2 \oplus e_5$ where $p_j \models e_5$ but $p_j \not\models e_2$:

- $e_2 \precsim e_4$
- $e_3 \precsim e_5$

VI. Theorems

The first theorem of this section allows to build the canonic form of the generalized of the example of the introduction. Let's consider the following conventions:

- The set of conflicts $(e_1 \perp e_2) \oplus (e_2 \perp e_3)...$ can be aggregated with

$$C = (\perp e_1\ e_2\ e_2\ e_3\ e_4\ ...\ e_p)$$

- Lets note l_i the i^{th} element of a list l.
- if e_i is an element of the list l, let's note $indice(e_i)$ the position of e_i in l

The next definition defines two U_n and V_n union of lists where the possible successor of an element e_i is $l(indice(e_i)+2)$ either $l(indice(e_i)+3)$:

Definition 7: Let's consider that $n <= p$:

$$\begin{cases} U_0 = & e_1 \\ U_n = & U_n^1 \cup U_n^2 \\ U_n^1 = & l_{n+2} \cup U_{n+2}^2 \\ U_n^2 = & l_{n+3} \cup U_{n+3}^2 \end{cases}$$

U_n defines all the processes beginning par e_1.

$$\begin{cases} V_0 = & e_2 \\ V_n = & V_n^1 \cup V_n^2 \\ V_n^1 = & l_{n+2}^1 \cup V_{n+2}^2 \\ V_n^2 = & l_{n+3}^1 \cup V_{n+3}^2 \end{cases}$$

V_n defines all the processes beginning par e_2.

Theorem 3: If we consider the conflict

$$C = (\perp e_1\ e_2\ e_2\ e_3\ e_4\ ...\ e_p)$$

Then this conflict can be canonically rewritten with the following form

$$C \equiv U_n \cup V_n$$

Proof: Correctness

Let's consider an incorrect process $q \in L_p$.

$$q = (\oplus\ e_{q1}\ e_{q2}\ e_{q3}\ e_{q4}\ ...\ e_{qp})$$

This incorrectness implies the existence of two events in q such as $e_{qi} \perp e_{qi+1}$ and e_{qi}, e_{qi+1} corresponding to two successive events of l. This is in contradiction with the definition of the functions $(U_n^1, U_n^2, V_n^1, V_n^2)$ for which events are added with either l_{n+2} either l_{n+3}. So the indices cannot be consecutive.

Completeness

Let's consider a valid process

$$q = (\oplus\ e_{q1}\ e_{q2}\ e_{q3}\ e_{q4}\ ...\ e_{qp})$$

not included in L_p. $\forall e \in q$, if q is valid then $\forall(e_i, e_j) \in q, \neg(e_i \perp e_j)$ so it implies that e_i and e_i are not successive in l and every enabled event is in q. Moreover, as q is not included in L_p, it exist at least one couple (e_{qi}, e_{qij}) which does not correspond to the construction defined by the functions $(U_n^1, U_n^2, V_n^1, V_n^2)$ which define the possible successor of an event. This means that $indice(e_{qj}) > indice(e_{qi} + 3)$. For every $n = indice(e_{qj}) - indice(e_{qi})$ greater than 3, lets note $i2 = indice(qi) + 2$ the event e_{qi2} is an possible event which is not in q (contradiction). ■

The following theorem is evoked in the introduction (section II), allows to compute maximal process:

Theorem 4: Let E an EAU, $e \in \mathcal{EB}$:

$$e \perp (e \oplus E) \equiv e \oplus E$$

Proof:

$$\begin{aligned} e \perp (e \oplus E) \quad &\equiv \quad (e \oplus t) \perp (e \oplus E) \\ &\equiv_{dist} \quad e \oplus (t \perp E) \\ &\equiv_{prop\perp} \quad e \oplus E \end{aligned}$$

■

The following theorem expresses how to develop a conflict:
Theorem 5 (Conflict):

$$e_1 \preccurlyeq (e_2 \perp e_3) \equiv (e_1 \preccurlyeq (e_2 \oplus \overline{e_3})) \perp (e_1 \preccurlyeq (\overline{e_2} \oplus e_3))$$

Proof:

$$\begin{aligned} e_1 \preccurlyeq (e_2 \perp e_3) \quad &\equiv_{def} \quad e_1 \preccurlyeq ((e_2 \oplus \overline{e_3}) \perp (\overline{e_2} \oplus e_3) \\ &\equiv_{dist} \quad (e_1 \preccurlyeq (e_2 \oplus \overline{e_3})) \perp (e_1 \preccurlyeq (\overline{e_2} \oplus e_3)) \end{aligned}$$

■

VII. Application

In the following example we show how to translate an unfolding and how to produce reduction to obtain the maximal processes of the unfolding. Let's considerer the Petri net in the figure 4:

The unfolding builds a table that explicit every relations between events and conditions, its representation is given in the figure 5:

This net is unsafe, P_3 may have 2 tokens, this appears in the unfolding with the two parts of the figure which distinguish between the token coming from P_1 and those from P_2.

The algebraic translation is produced with the following transformations:

- Each arc is a causal relation $e_1 \preccurlyeq e_2$
- Each conflict:

$$e \preccurlyeq (\perp \ e_1 \ e_2 \ \dots \ e_n)$$

- When more than one condition b_i is expressed on an event:

$$(\oplus \ b_1 \ b_2 \ \dots \ b_n) \preccurlyeq e$$

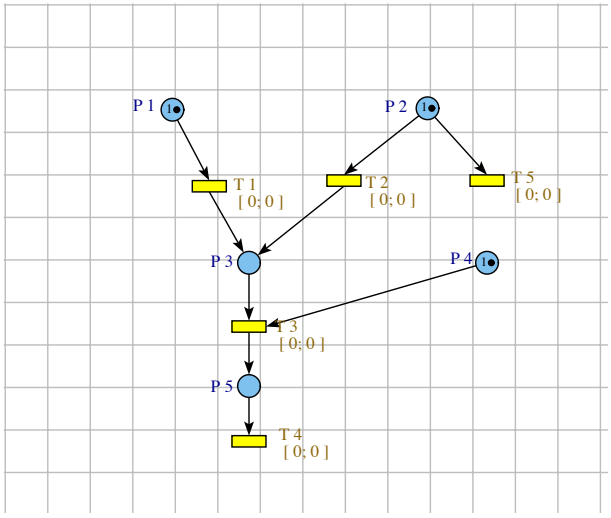

Fig. 4. Petri net

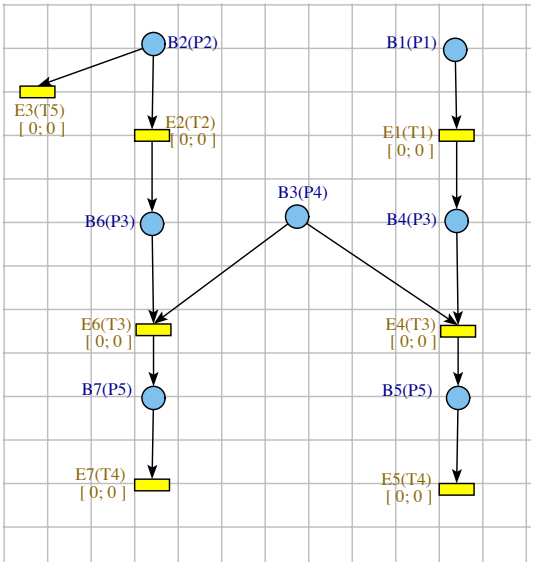

Fig. 5. Unfolding

- $Min(\mathcal{O})$, the initial marking:

$$\oplus \ b_1 \ b_2 \ \dots \ b_n$$

The unfolding is expressed by:

$$\begin{aligned} U = \quad & (\oplus \ b_1 \ b_2 \ b_3 \quad \text{(Initial marking)} \\ & (b_2 \preccurlyeq (e_3 \perp e_2)) \\ & (e_2 \preccurlyeq b_6) \\ & ((b_3 \oplus b_6) \preccurlyeq e_6) \\ & (e_6 \preccurlyeq b_7) \\ & (b_7 \preccurlyeq e_7) \\ & (b_1 \preccurlyeq e_1) \\ & (b_3 \preccurlyeq (e_6 \perp e_4)) \\ & (e_1 \preccurlyeq b_4) \end{aligned}$$

$$((b_3 \oplus b_4) \preccurlyeq e_4)$$
$$(e_4 \preccurlyeq b_5)$$
$$(b_5 \preccurlyeq e_5))$$

Let's note

$$U = (\oplus\ b_2\ (b_2 \preccurlyeq (\perp\ e_3\ e_2))\ P)$$

The derivation rule D is applied on:

$$\oplus\ b_2\ (b_2 \preccurlyeq (\perp\ e_3 e_2))$$

and U becomes:

$$U = \oplus\ (\perp\ (\oplus\ e_3\ \bar{e_2})\ (\oplus\ \bar{e_3}\ e_2))\ P$$

The distributivity of $\oplus$ on $\perp$, allows to exhibit two processes:

$$\perp\ (\oplus\ e_3\ \bar{e_2}\ P)(\oplus\ e_2\ \bar{e_3}\ P)$$

Let's note in the following

$$P_1 = (\oplus\ e_3\ \bar{e_2}\ P)$$

and

$$P_2 = (\oplus\ e_2\ \bar{e_3}\ P)$$

A term of P is $e_2 \preccurlyeq b_6$. Let's define P' as

$$P = (e_2 \preccurlyeq b_6)\ P'$$

Thus:

$$P_1 = \oplus\ e_3\ \bar{e_2}\ P = \oplus\ e_3\ \bar{e_2}\ (e_2\ \prec\ b_6)\ P'$$

The MP' simplification can be done:

$$P_1 = \oplus\ e_3\ \bar{e_2}\ (e_2 \preccurlyeq b_6)\ P' \Rightarrow \oplus\ e_3\ \bar{b_6}\ P'$$

Let's note:

$$P_{1a} = \oplus\ e_3\ \bar{b_6}\ P'$$

In the same manner, let's note P'' as

$$P' = \oplus((\oplus\ b_3\ b_6) \preccurlyeq\ e_6)\ P''$$

Thus:

$$P_{1a} = \oplus\ e_3\ \bar{b_6}\ ((\oplus\ b_3\ b_6) \preccurlyeq e_6)P''$$

The GMP' rule gives P_{1a}:

$$P_{1a} \Rightarrow e_3 \oplus \bar{e_6} \oplus P''$$

Thus P_1 becomes:

$$\begin{aligned} P_1 \quad \Rightarrow \quad & \oplus\ b_1\ b_3\ e_3\ \bar{e_2}\ \bar{b_6}\ \bar{e_6}\ \bar{b_7}\ \bar{e_7} \\ & (b_3 \preccurlyeq (e_6 \perp e_4)) \\ & (b_1 \preccurlyeq e_1)) \\ & (e_1 \preccurlyeq b_4) \\ & ((\oplus\ b_3\ b_4) \preccurlyeq e_4) \\ & (e_4 \preccurlyeq b_5) \\ & (b_5 \preccurlyeq e_5) \end{aligned}$$

The same simplifications are applied twice to the left part of the unfolding: $\bar{e_6}$ treats the conflict $b_3 \preccurlyeq (e_6 \perp e_4)$ and P_1 becomes:

$$P_1 \Rightarrow \oplus\ e_3\ \bar{e_2}\ \bar{b_6}\ \bar{e_6}\ \bar{b_7}\ \bar{e_7}\ e_1\ b_4\ e_4\ b_5\ e_5$$

In this process, e_3 corresponds to the firing of T_5, this process illustrates the case where the activity of the net comes from the token in P_1. Another proof sequence (firing of T_2) can be conducted to lead to the two other processes. One for the case leading to the event e_5 and the other leading to the event e_7.

VIII. Implementation aspects

The unfolding has been implemented in Racket/Scheme [13] (a recent evolution of $LISP$ language). The unfolding produces an hashing table. The couples of events are associative keys for the table. For every couple of events, the relation is explicated as follow:

- $table[e_1, e_2] = t$ if $e_1 \preccurlyeq e_2$ ($\prec$ corresponds to the definition of section III-B) ;
- $table[e_1, e_2] = f$ if $e_1 \perp e_2$;
- if (e_1, e_2) is not a key of the table then $e_1 \wr e_2$.

The axiomatisation of the algebra allows to solve the previous example like a fixed-point formulae. When non more rules can be applied the final formulas (the leaves of the prove tree) gives the set of maximal processes.

The algebra and reduction rules have been defined with a package named "PLT/Redex" [12]. A $Redex$ Model is formed of two parts: a regular-tree grammar and reduction rules.

The following part of code is the grammar. It defines the Algebraic Expression of Unfolding AEU, the process P, X the conflicts (section 1) and E a well formed expression. The following code gives the $S - expression$ (programs, expressions, variables, or literal constants) and contexts which are parts of the syntax of the language.

```
(define-language Process-lang
  [n variable bool b e (¬ n)] ; node
  [bool t f] ; boolean
  [e variable (¬ e)] ; event
  [b variable (¬ b)] ; condition
  [on (⊕ ⊥)]  ; n-nary operators
  [o2 ≺]  ; binary operators
; process
 [P variable (⊕ Q  ...)]
  [Q variable P n]
  [C-P (⊕ C-P P) (⊕ P C-P) hole]
; Conflicts
 [X variable (⊥ Y ... )]
  [Y variable X n ]
  [C-X (⊥ C-X P) (⊥ P C-X) hole]
; Expression
  [E variable (on F ...) (b o2 e) (P o2 X)]
  [F variable E P ]
  [C-E (on C-E E) (on E C-E)
  (E o2 C-E) (C-E o2 E) hole]
)
```

The $define - language$ key-world specify the abstract syntax with a fully parenthesized notation:

- The second line defines the set of terms: e

- The following defines boolean type, the events and the conditions, then $n-ary$ operator and $unary$ operator.
- Then comes the definitions of process (P), conflicts (X) and Expression.
- The keyword $variable$ means that a variable can be instantiated: for example e can be instantiated by $e_1, e_2, ...$
- The ellipse $F...$ represent the regular expression F^*
- Finally, $C-P$, $C-X$ and $C-E$ are respectively the contexts of the processes, the conflicts and the expressions.

This second part describes pattern-based formulation of reduction possibly in context.

```
(define Process-red
  (reduction-relation
   Process-lang
;-------------------------------------------------------
; (⊕
  ; Absorbing element
 (--> (in-hole C-P (⊕ Q_1 ... f Q_2 ... ))
      (in-hole C-P f))
 ;-------------------------------------------------------
; Falsification
 (--> (in-hole C-E (⊕ Q_1 ... e_1 Q_2 ... (¬e_1) Q_4 ...))
     (in-hole C-E f)  )
 ;-------------------------------------------------------
; associativity
(--> (in-hole C-E (⊕ E_1 ... (⊕ E_2 ...) E_3 ...))
        (in-hole C-E (⊕ E_1 ... E_2  ...  E_3 ...) ))
;-------------------------------------------------------
; Distributivity of ⊕ over ≺
 (--> (in-hole C-E (⊕ E_1 ... (≺ E_2 E_3) E_4 ...))
        (in-hole C-E (≺ (⊕ E_1 ... E_2 E_4 ...)
        (⊕ E_1 ... E_3 E_4 ...) )))

 ;-------------------------------------------------------
; Conflict between 2 events
(--> (in-hole C-E (⊕ b ... E_1 ...
                    ((⊕ b ...) ≺ (⊥ n_1 n_2)) E_2 ...))
       (in-hole C-E (⊥ (⊕ n_1 (¬n_2) E_1 ... E_2 ...)
                       (⊕ n_2 (¬n_1) E_1 ... E_2 ...))))
 ;-------------------------------------------------------
; Conflict between n events
 (--> (in-hole C-E (⊥ e ...)) (in-hole C-E (f (⊥ e ... ))))
;-------------------------------------------------------
; ≺: Causality
   ((⊕ b_0 ... ((b_1 ≺  e) E ...)) . --> .
       ( (⊕ b_0 ... ((b_1 ≺  e) E ...)))"Caus_S")

   ((⊕ b_0 ... ((⊕ b_1 ...) ≺ e) E ...) . --> .
        ( (⊕ b_0 ... ((⊕ b_1 ...) ≺ e) E ...))"Caus_G")
))
;-------------------------------------------------------
; META FUNCTION
(define-metafunction Process-lang
;  [( (⊕n ...)) ,(cons '⊕(symbol-sort (term (n ...))))]

  [( (⊕ b_0 ... (b_1 ≺  e) E ...))
   ,(cons '⊕(and (member (term b_1) (term (b_0 ...)))
                 (append (symbol-sort (append
                        (complement (term (b_0 ...))
                                    (term b_1 ))
                        (list (term e)))
                        (term (E ...)))))))]

  [( (⊕  b_0 ...  ((⊕ b_1 ...) ≺  e) E ... ))
   ,(cons '⊕(and (? (term (b_1 ...)) (term (b_0 ...)))
                 (append (symbol-sort (append
                        (complement (term (b_0 ...))
                                    (term (b_1 ...)))
                        (list (term e))))
                        (term (E ...))))))]
   [( (⊥ e ... )) ,(lproc (term (e ...)) ))  ]   )
```

The first part express basics properties: absorbing element and associativity of $\oplus$. The second part illustrates how causality and conflicts are reduced.

f is a meta-function which has several rules: It is first used to ease the resolution of conflicts:

- The first item sorts the symbol in alphabetic order. It is useful to assure the canonicity of the expression;
- The second verifies that if b_1 is member of the sequence $b_0...$ then b_1 is substituted by e.
- The third verifies that if the sequence $b_1..$ is a subsequence of $b_0...$ then $b_1...$ is substituted by e.

The second rule of f is to implement the computing of

$$L_p = U_n \cup V_n$$

The auxiliary function $lproc$ is the implementation of the theorem (3). f builds for every list of event in conflicts of type:

$$(\perp\ e_1 ... e_n)$$

the set of resulting processes.

A. Example 1

Let's consider the examples given in the introduction of the paper, if we apply the reduction rules defined in our algebra of the first graph, 4 maximal processes expressed in canonical form are obtained:

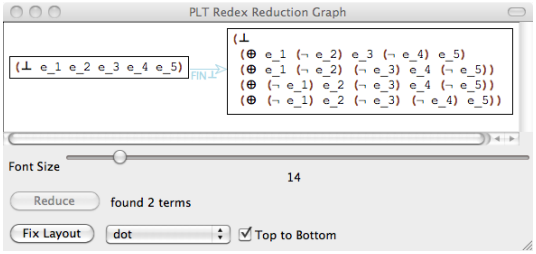

Fig. 6. $(\perp\ e_1\ e_2\ e_3\ e_4\ e_5)$

Extraction of semantics: The canonical form of this expression has been obtained in one pass by this implementation of the theorem 3.

$$\begin{aligned}(\perp\ e_1\ e_2\ e_3\ e_4\ e_5) = (\perp\quad & (\oplus\ e_1\ \overline{e_2}\ e_3\ \overline{e_4}\ e_5)\\ & (\oplus\ e_1\overline{e_2}\ \overline{e_3}\ e_4\ \overline{e_5})\\ & (\oplus\ \overline{e_1}\ e_2\ \overline{e_3}\ e_4)\\ & (\oplus\ \overline{e_1}\ e_2\ \overline{e_3}\ e_5))\end{aligned}$$

Theses processes exhibits two groups of causalities: $(e_1 e_3 e_5)$ and $(e_2 e_4)$. Moreover the following relations can be deduced:

- $e_3 \preccurlyeq (e_1 \oplus e_5)$,
- $(e_1 \oplus e_3) \preccurlyeq e_5$
- $(e_1 \oplus e_5) \preccurlyeq e_3$
- $(e_3 \oplus e_5) \preccurlyeq e_1$, ...
- $e_1 \precsim e_5$, $e_3 \precsim e_5$, ...
- $e_2 \precsim e_4$, $e_4 \precsim e_2$,...

In the second form:

$$(\perp\ e_6\ e_7\ e_8) \equiv ((\oplus\ e_6\ \overline{e_7}\ e_8) \perp\ (\oplus\ \overline{e_6}\ e_7\ \overline{e_8}))$$

It can be deduced that $e_6 \preccurlyeq e_8$ and $e_8 \preccurlyeq e_6$

B. Example 2

Concerning the Petri Net presented in this section, the analyze of the unfolding gives 138 applications of derivation rules and theorem. It finally exhibits four maximal processes in canonical form which are the leaves of the demonstration tree (two frames from which where none rules can be applied any more):

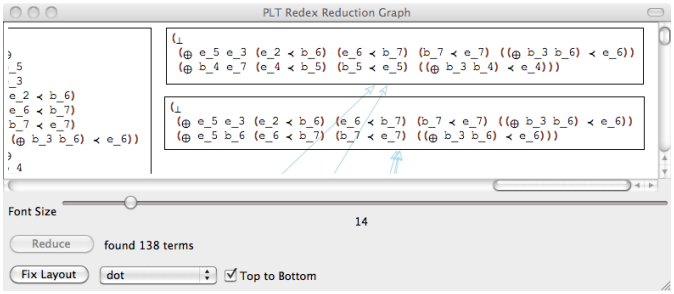

Fig. 7. Unfolding of figure 4

Extraction of semantics: : The information that can be extracted of the analyze is:

- If e_7 occurs that the condition b_4 is pending.
- If e_5 occurs and e_3 does not then the condition b_4 is pending.

The definitions of $\precsim$ and $\preccurlyeq$ allow to extract the following relations: In each process p_i, we can verify that either $\overline{e_3}$ belongs to p_i either $(\oplus\ e_3\ e_5)$ either $(\oplus\ \overline{e_3}\ e_5)$. So we can deduces that $e_3 \precsim e_5$.

Moreover, $\forall p \in U$, when $b_4 \models p$ then $e_7 \models p$. This induces that $b_4 \preccurlyeq e_7$. In the same manner we can prove that $b_6 \preccurlyeq e_5$,...

IX. Conclusion

This work is a first attempt to establish an algebra for reasoning about branching process. It shows that an unfolding of a Petri net can be represented in an algebraic way. This algebra leads to express the unfolding in a canonical form.

This algebraic framework has been implemented in $PTL/Redex$, which is a module based on the language $Racket$ (last evolution of $LISP$).

The canonical form is induced by the structure of the algebra and the consideration of an alphanumeric sort on the symbols. This form illustrates processes that are in exclusion. From this form, non intuitive relations can be established.

Some perspectives of this work can be evoked. First, the procedure of extraction of semantics needs to be automatized and other types of relations between events need to be explored.

This paper propose an analyze of the unfolding of non-temporized Petri net. A second perspective, could be to extend this work to temporized Petri nets. In $PLT/redex$, some works shows how to introduce rules to extends models with the axiomatics calculus on integer (based on $\lambda_{calculus}$). Theses adding rules could help to manage the computing of temporalities.

References

[1] Bernard Berthomieu and François Vernadat. State class constructions for branching analysis of time Petri nets. In *TACAS*, pages 442–457, 2003.

[2] Jean Yves Girard. Linear logic, Theorical Computer Science, 1987, volume = 50

[3] Thomas Chatain and Claude Jard. Complete finite prefixes of symbolic unfoldings of safe time Petri nets. *Springer-Verlag*, 4024:125–145, 2006.

[4] Javier Esparza and Keijo Heljanko. *Unfoldings: A Partial-Order Approach to Model Checking (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[5] Joost Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28(6):575– 591, june 1991.

[6] Javier Esparza, Stefan Römer, and Walter Vogler. An improvement of McMillan's unfolding algorithm. In Tiziana Margaria and Bernhard Steffen, editors, *TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 87–106. Springer, 1996.

[7] Keneth L. McMillan. Using unfolding to avoid the state explosion problem in the verification of asynchronous circuits. In *Proc. Fourth Workshop on Computer-Aided Verification*, volume 663, pages 164–177, Montreal, Canada, june 1992. Springer-Verlag.

[8] Keneth L. McMillan. A technique of state space search based on unfolding. *Form. Methods Syst. Des.*, 6(1):45–65, 1995.

[9] Philip Meir Merlin and David. J. Farber. Recoverability of communication protocol - implications of theoretical study. *IEEE Transactions on Communications*, 24:1036–1043, September 1976.

[10] Robin Milner A Calculus of Communicating Systems *Lecture Notes in Computer Science*, Springer-Verlag, 1980.

[11] Pierre Wolper and Patrice Godefroid. Partial-order methods for temporal verification. In Eike Best, editor, *CONCUR*, volume 715 of *Lecture Notes in Computer Science*, pages 233–246. Springer, 1993.

[12] Matthias Felleisen, Robert Bruce Findler and Matthew Flatt Semantics Engineering with PLT Redex MIT Press

[13] Gerald Jay Sussman and Guy Lewis Steele, Jr.. Scheme: An Interpreter for Extended Lambda Calculus MIT AI Lab. AI Lab Memo AIM-349. December 1975.